L2/02-379

Title: Topics for Character Set Ad Hoc
Date: 2002-08-12
From: Frank Farance

This document discusses the following topics.

- An Overview of "Characters"
- Using Characters
- Designators Based On International Characters
- Character Naming, Datatyping, and Representation
- Internationalization and Localization Issues for Characters

Topic: An Overview of "Characters"

Regarding this topic, I suggest that SC22 develop, in collaboration with other JTC1 SCs and with consortia (e.g., Unicode, W3C, IETF), a short Type 3 Technical Report on an overview of "characters" and there use in information technology. This TR would compile excerpts from a variety of *existing* sources. For example, the following topics would be covered:

- Terminology: character, coded character, character set, script, etc.. Note: much of this would be taken from 10646-1.
- Resources/Documentation: URLs to important documents that are relevant (e.g., SC2 documents, SC22/WG20 documents, Unicode documents, W3C documents, IETF documents).
- Design/Implementation Issues: answers the question: What choices do I have? This would be a good place to discuss some of the typical issues that concern developers/users. For example, choosing among representations (pros/cons for some common representations), fixed length vs. variable length character representations, portability issues (e.g., why it might be useful to write `"\uhhhhh"` rather than inserting the character directly), escaping/ tunneling mechanisms (how to count the right number of backslashes or using `"&#xxxx;"` properly), format vs. markup vs. text, choosing the formulae for determining "smallest permitted maxima" (a.k.a., minimum-maxima) when writing specifications, locale-specific behavior.
- Code samples: some samples of code that works the same across several platforms/environments, and some samples of code that interoperate (note the former is not the same as the latter).
- Frequently Ask Questions: Answers to the FAQs about characters (see the rest of this document for potential topics).

Additionally, wording from the following topics below could be incorporated into the TR.

Topic: Using Characters

In this topic, I believe that both "character processing" and "character properties" are included. In SC22 standards, we use characters (non-exhaustive list):

- For input and output to user interfaces, storage, communication systems, etc.
- For spelling expressions and statements in our programming languages
- For spelling "keywords" in our programming languages
- For spelling literal values of a variety of datatypes
- For spelling common methods of data interchange, e.g., a text (non-binary) representation of an integer
- For spelling the designators associated with external objects, such as files and communication system endpoints
- For spelling the designators associated with internal objects (a.k.a., "identifiers")
- For spelling natural language, whether it is comments embedded in programs, it is string literals, or it is processing intermediate values
- For control functions and formatting functions

A variety of paradigms exist for programs that process/use characters. One paradigm, let's call it "character services", focuses on the "services" (e.g., string concatenation, searching, conversion, etc.) provided by the language/environment. Another paradigm, let's call it "character properties", focuses on providing a rich set of inquiry capabilities regarding the characteristics of characters (e.g., "is-latin", "is-graphic", "is-digit", "is-math"). Both of these paradigms exist in programming languages.

Additional paradigms concerns words (word-based paradigms), lines (think UNIX), and "paragraphs" (think DOM and various markup languages).

SC22 should describe these paradigms in a language-independent way so that there is some consistency across programming languages. The need for consistency is: programmers in one language abstract and build, based on the features of their language. It took a long while to harmonize the lines vs. records paradigm (think Fortran and C), but I believe the results have been worthwhile.

Of course, much of the discussion and rationale for this kind of work should be incorporated into the TR above.

Topic: Designators Based On International Characters

Concerning the 10176 activities, it would be appropriate for the following to happen:

- Administering the maintenance were handled long-term by a single organization (e.g., Unicode, WG20, etc.)

- Regardless of where the maintenance is administered, SC22 WGs and SC32 WGs (WG1, WG2, WG3) do need review of any changes -- for example, there might be discussion of whether or not characters such as "(0)" (zero with a circle) are considered valid identifier characters.
- The 10176 table should separate the table into "letters-like characters", "digit-like characters", and "other characters" so programming and database languages can properly extend their syntax for letter-like things and for digit-like things (common lexical categories). The choice of whether the table is actually split or attributes are added to table entries is merely a presentation issue.

Again, some of this discussion should be added to the TR.

Topic: Character Naming, Datatyping, and Representation

Regardless of what it says in 10646, people seem to be unaware that it specifies the naming (i.e., referencing) characters — in fact, several ways of naming characters (e.g., **`"U-00000028"`** vs. **`"LEFT PARENTHESIS"`**). The C/C++, IETF, and W3C HTML/XML approaches should be highlighted:

- How C/C++ names characters (note that these are representation-independent)
- IETF (note IETF's use of UTF-8)
- W3C (note its use of hex and decimal names; note its use of other names, such as **`"&copy;"`**)

So the result of this work might be a language-independent and representation-independent way of describing the characters (yes, this might just point back to 10646, but one needs to show how 10646 is applied in this context), including the use of non-hex (and non-decimal) based names.

Regarding datatyping, much work has been done in C to have datatypes that are portable, yet efficient across many environments. The same approach should be taken here: We need a datatype for a "character", yet one supports various "ranges". For example, C's **`"unsigned char"`** should be useful for character sets that have up to 256 codes (other synonyms might be useful, too). For C's wide character datatype, there is no requirement that it support anything larger than 8 bits. So possibly 2 more datatypes are necessary: a datatype that supports at least 2^16 codes, and a datatype that supports at least 2^32 codes. Please note that C's **`<stdint.h>`** provides both "at least" and "exact" datatypes, so the corresponding datatypes would be useful for characters (i.e., characters of "at least 16 bits" and "exactly 16 bits"). Note: This is not a recommendation for providing all the support services for each of these datatypes, just a proposal for providing the raw datatype.

Regarding representation, programming languages need to describe in a common way the representation of characters for, say, I/O stream processing. Rather than each programming language devising its own scheme, some common techniques should be applied. For example, one might need to know the kind of data and its encoding when specifying a stream.

IETF has a good number of RFCs in this regard and I suggest that they are incorporated for reuse (e.g., **`Content-Type: text/html; charset="iso-8859-1"`**). These could easily be incorporated into existing APIs:

```
fopen("file","r,Content-Type: text/html; charset=\"iso-8859-1\"");
```

Again, some of this discussion should be added to the TR, including existing standards (SC22, SC2, IETF, W3C, etc.).

Topic: Internationalization and Localization Issues for Characters

Several SC22 WGs have done work on internationalization features (e.g., sorting, printing). And private companies (e.g., IBM) and consortia (e.g., Unicode) have done work. SC22 should investigate these existing services and determine what can be standardized. It is expected that all work would be language independent in nature; some work may be representation independent, while other work might be representation dependent (e.g., UTF-8, UTF-16).

Again, some of this discussion should be added to the TR.